# A Risk-based Approach to Strategic Decision-Making for Software Development

James D. Kiper
*Department of Computer Science and Systems Analysis*
*Miami University*
*Oxford, OH 45056*
*kiperjd@muohio.edu*

Martin S. Feather
*Jet Propulsion Laboratory*
*California Institute of Technology*
*4800 Oak Grove Dr*
*Pasadena CA 91109-8099*
*Martin.S.Feather@Jpl.Nasa.Gov*

**Abstract**

*In any software system development, the most important strategic decisions are, by definition, those made early in the lifecycle. However, these early lifecycle decisions are generally made in a data-starved environment. The best sources of data are those based on historical information (if the current project is sufficiently similar to past systems), and the judgments of domain experts.*

*At NASA, we have been developing and applying a risk-based model to capture information from domain experts and to study and plan for systems that use advanced technology.*

*Here we describe the "Defect Detection and Prevention" (DDP) model and software tool. This model and the custom built tool that implements it initially arose from needs in the hardware domain. However, current spacecraft systems are a complex combination of hardware and software. In this paper, we describe some initial work investigating the applicability of this model and tool to software components.*

## 1. Introduction

Strategic decision-making for software development of necessity occurs early in the project lifecycle when data is scarce. However, these strategic decisions are the most critical ones in the entire development process since they constrain the set of possible future, more detailed, decisions that can be taken. Two common approaches to finding data on which to base such early decisions is to use historical data from similar projects, and to based decisions on the judgments of domain experts. In NASA applications to spacecraft and related advanced technology, there are often novel aspects to the missions that require a combination of both historical data and expert judgment of engineers. Fortunately, NASA is blessed with a large number of world-class engineers. In this paper we focus primarily on utilizing their expert judgment.

The "Defect Detection and Prevention" (DDP) model and software tool have been developed and are currently being used at the Jet Propulsion Laboratory (JPL) to capture early lifecycle decisions from engineering experts which is then used to create plans to meet project cost, schedule, and quality goals. The model that underlies DDP is a risk-based one. The model's elements are described in section 2 and its calculations in section 3. (These descriptions are adapted from that of [3].) The DDP process [6] is the subject of section 4. DDP initially was designed to meet a need for strategic planning of hardware components of systems. This is still its primary use at JPL. However, current spacecraft are a complex combination of hardware and software systems that work together to accomplish mission goals. In section 5 we present a proof-of-concept exercise that illustrates how DDP might be applied to software.

## 2. DDP Risk Model

The simple quantitative model at the heart of DDP involves just three key concepts: "*Requirements*" (what it is that the system or technology is to achieve), "*Risks*" (what could occur to impede the attainment of the Requirements), and "*Mitigations*" (what could be done to reduce the likelihood and/or impact of Risks). Requirements are related to Risks, and Risks are in turn related to Mitigations. Specifically, Requirements are quantitatively related to Risks to indicate how much each Risk, should it occur, *impacts* each Requirement. Risks are quantitatively related to Mitigations, to indicate how much of a Risk-reducing *effect* a Mitigation, should it be applied, has on reducing each Risk.

The subsections that follow give the details of DDP's key concepts: Requirements, Risks and Mitigations, and the Impact and Effect relationships between them.

### 2.1 Requirements

*Requirements* are whatever the system under scrutiny is to achieve, and the constraints under which it must operate. They can be "product" requirements on the system (e.g., functional behavior, run-time resource needs, timing requirements), and/or "process" requirements (on the development process itself, e.g., development environment, testing facilities, progress reporting requirements). Each requirement is assigned a *weight*, representing its relative importance to mission success, etc.

## 2.2 Risks

*Risks* are all the things that, should they occur, will lead to loss of Requirements. Each Risk is assigned an *a-priori likelihood* (the chance of the Risk occurring, if nothing is done to inhibit it). Each Risk is also assigned a *repair cost*, what it would cost to remove an instance of that Risk from the system. The DDP model allows for a distinction to be made among various time phases; in that case, the repair cost may be different for each of those possible time phases.

## 2.3 Mitigations

Mitigations are all the activities that could be done to reduce the likelihood of Risks and/or reduce their impact on Requirements. These include *preventative measures* (e.g., training, standards, selection of high quality parts), *detections* that discover instances of Risks through analysis or test (e.g., code walkthroughs) so that those detected Risks can be corrected prior to release/use, and *alleviations* that reduce the severity of Risks (e.g., defensive programming that checks its inputs to ensure they are within specified bounds). We henceforth refer to these different kinds of mitigations as "prevention", "detection" and "alleviation" mitigations.

Each mitigation is assigned a *cost*, the cost of performing it. Cost may be a measure of budget, schedule, physical attributes (e.g., weight and electrical power are predominant concerns for spacecraft), scarce resources (e.g., skilled personnel, high fidelity testbeds), or indeed a mixture of these measurements. Each mitigation is also assigned the time period within the development effort at which it would be performed (e.g., requirements phase, design phase).

It is possible that a mitigation can *induce* a Risk. For example, inserting error detection code can change the run-time behavior of a system, and thus increase the risk of timing errors.

## 2.4 Impacts

For each Requirement x Risk pair, the "impact" is the proportion of the Requirement that would be lost if the Risk were to occur. It is expressed as a number in the range 0 – 1, where 0 means no impact whatsoever, and 1 means total loss of the Requirements. Note that a Risk may impact multiple Requirements and do so to differing extents. Likewise, multiple Risks may impact a Requirement, again to differing extents.

Impacts combine *additively*, e.g., if two different Risks impact the same Requirement, then their combined impact on that Requirement is calculated as the sum of their individual impacts.

One seemingly strange consequence of our combination rule for impacts is that a Requirement can be *more* than completely impacted! For example, impacts of 0.8 and 0.7 on the same requirement add up to a total impact of 1.5. This is in fact a useful measure, of the amount of risk to be overcome in order to attain the requirement. However, when assessing how much of the Requirements have actually been attained, Requirements that are more than completely impacted contribute zero (not a negative amount, note).

The usual notion of risk is a triple – its identity, likelihood and consequence. Identity and likelihood are provided in the definition of the risk, while consequence is derived as the sum total impacts the risk has on objectives.

## 2.5 Effects

For each Mitigation x Risk pair, the *Effect* is the proportion by which that Risk would be reduced if that mitigation were applied. It is expressed as a number in the range 0 – 1, where 0 means no reduction whatsoever, and 1 means total elimination of the Risk.

Effects combine "*multiplicatively*": when several

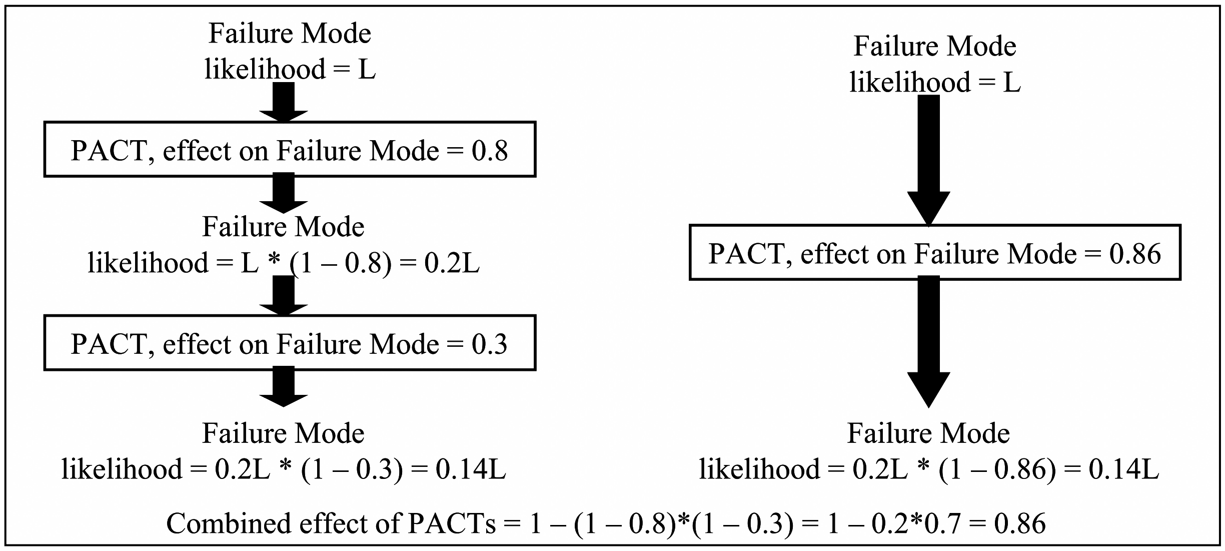

Failure Mode
likelihood = L
PACT, effect on Failure Mode = 0.8
Failure Mode
likelihood = L * (1 – 0.8) = 0.2L
PACT, effect on Failure Mode = 0.3
Failure Mode
likelihood = 0.2L * (1 – 0.3) = 0.14L
Failure Mode
likelihood = L
PACT, effect on Failure Mode = 0.86
Failure Mode
likelihood = 0.2L * (1 – 0.86) = 0.14L
Combined effect of PACTs = 1 – (1 – 0.8)*(1 – 0.3) = 1 – 0.2*0.7 = 0.86

**Fig. 1.** DDP PACTs act like "filters" in series

mitigations reduce the same Risk, their combined effect is computed as: (1 – the product, for each Mitigation M, of (1 – M's effect)).

Intuitively, mitigations act as "filters" arranged in series, such that each mitigation filters out its effect's proportion of the Risks that enter it. See Figure 1 for an example in which a mitigation with an effect of 0.8 on some Risk and another mitigation with an effect of 0.3 on that same Risk are each applied.

Note that the order in which these mitigations are applied does not matter.

As was the case for impacts, a mitigation may affect multiple Risks and do so to different extents, and a Risk may be "affected" by multiple mitigations, again to different extents.

Mitigations that *induce* Risks are taken into account by having them *increase* the likelihood of those Risks. Again, the degree of this influence is expressed as a number in the range 0 – 1. For a Risk with likelihood L, and mitigation with inducing effect of E, the new likelihood is calculated as (1 – (1-L)*(1-E)). Intuitively, if the Risk was going to occur anyway or it is induced by the mitigation (or both), then it will occur. Since the likelihood of (P1 or P2) = (1 – (Likelihood of P1) * (1 – Likelihood of P2)), we get the formula above. Thus, at the extremes for the mitigation's inducing effect E, 0 means no increase, and 1 means increase to certainty.

For example, if L=0.4 and E=0.7, this calculation is: (1 – (1-0.4)*(1-0.7)) = (1 – 0.6*0.3) = 0.82.

Note the order in which Risk *reducing* mitigations interleave with Risks *inducing* mitigations *does* matter. For example, consider a "perfect" mitigation (one that reduces a Risk's likelihood to 0) and a Risk inducing mitigation. If the perfect mitigation follows the inducing one, the Risk will be eliminated, while the other way around, the inducing mitigation will cause the Risk to occur after the point at which the perfect has had a chance to apply. In practice, we assign mitigations to distinct time phases, and organize the calculations so that for mitigations of a given phase, all the likelihood-reducing effects are calculated first (the relative order of which does not matter), and all the likelihood-increasing effects are calculated second (again, the relative order of which does not matter). This means that the Risks induced within a time phase can be reduced only by mitigations of later time phases.

It has been suggested that, when possible, the ordering of mitigations could be deliberately chosen to optimize their net effect. For example, given two mitigations that could be applied in either order, chose the ordering that puts first the mitigation that induces Risks. This would be an interesting extension to the current DDP implementation.

## 3. DDP Calculations

In a DDP model, a set of Mitigations achieves *benefits* (Requirements are met because the Risks that impact them are reduced by the selected Mitigations), but incurs *costs* (the sum total cost of performing those Mitigations).

The measure of benefit of a DDP model is calculated as the sum of the weighted requirements' attainment. The measure of cost of a DDP model is calculated as the sum of the costs of the Mitigations selected for application, plus the sum of the costs of repairs of the Risks that detection mitigations discover. Both of these measures take into account the detrimental impact of Risks on Requirements, moderated by the effect of Mitigations at reducing Risks' likelihoods and/or severities.

The essential aspects are the calculation of Risks' likelihoods and severities (in the course of which costs of mitigations and repairs are accumulated), followed by the calculation of Requirements' attainment. These are described next.

### 3.1 Risk Likelihoods and Severities

The calculation of each Risk's likelihood starts from its a-priori likelihood value. At each time phase, the effects on it of that phase's prevention and reduction mitigations reduce its likelihood. As discussed earlier, a Mitigation acts as a "filter" to remove some proportion of the Risk. In the course of this calculation, the reduction in likelihood attributable to detection mitigations incurs a repair cost. This is the repair cost attributed to the Risk at that phase, multiplied by the proportion by which the Mitigation reduces the Risk's likelihood.

EXAMPLE: consider a Risk (e.g., a requirements flaw) that costs $100 to repair at requirements formulation time. Suppose a Mitigation (e.g., requirements inspection) has an effectiveness of 0.7 against that Risk. If the Risk's likelihood prior to application of the mitigation is 0.9, then after it will be 0.9 * (1 – 0.7) = 0.27. The reduction in likelihood is 0.9 – 0.27 = 0.63, and so the repair cost is $100 * 0.63 = $63. An equivalent and more direct calculation of this is to simply multiply the Risk's unit repair cost ($100) by its likelihood prior to mitigation (0.9) by the mitigation's effect on that Risk (0.7): $100 * 0.9 * 0.7 = $63.

The mitigations of a time phase that *induce* Risks are taken into account after all the mitigations of that phase that reduce Risks. Their contribution is calculated using the combination rule discussed in the Effect subsection earlier.

The severity reductions attributable to alleviation mitigations are also calculated phase by phase, using the same kind of calculation as prevention mitigations, but decreasing Risk severities rather then likelihoods.

### 3.2 Requirements Attainment

The ideal requirements attainment is simply the sum of the weights of all the requirements. This ideal would only be achieved if all the Risks were completely mitigated, by reducing their likelihoods and/or severities to zero.

The actual attainment of a requirement, taking into account Risks and mitigations, is its weight * (1 – the proportion to which it is at risk, capped at 1). The proportion to which it is at risk is the sum over all Risks of each Risk's (likelihood * severity * impact on that Requirement). As mentioned earlier, this sum can exceed 1, hence the need to cap it at 1 in this calculation. The Risks' likelihoods and severities are calculated as described in the previous subsection.

EXAMPLE: consider a requirement with weight 100 that is at risk due to two Risks. Suppose that after taking mitigations into account, the first Risk has likelihood 0.9, severity 0.5 and impact 0.5, and the second has likelihood 0.4, severity 1.0 and impact 0.3. This requirement's attainment is thus (100 * (1 - ((0.9 * 0.5 * 0.5) + (0.4 * 1.0 * 0.3)))) = (100 * (1 - (0.225 + 0.12))) = (100 * (1 - 0.345)) = 65.5.

### 3.3 Ontology Complexity

The DDP modeling *language* is fairly simple – consisting of three first class entities (Requirements, Risks, and Mitigations) and relationships between requirements and risks, and between risks and mitigations. There are surely situations where this fairly simple ontology is not adequate to model complex relationships. However, for the early design of complex technology in which it is currently being used at JPL this simple ontology is an advantage. First, this allows us to extract a reasonably robust and accurate model from a group of very intelligent, but extremely busy engineers in a few (three or four), relatively short (half day) meetings. Since this model is used early in the design process, it can point engineers (in a fairly rudimentary way) to where problems are likely to occur. Then a more complex and complete model can be used to analyze these problem areas more thoroughly.

## 4. DDP Process

The success of a DDP application is crucially dependent on the involvement of experts. Their combined expertise must encompass:

Requirements:

- Driving needs/goals/objectives (e.g., in our setting, the science mission objectives driving the need for an instrument's capabilities).
- Environmental constraints on resources available to the system (e.g., RAM, power).
- Environmental constraints on the extent to which the system can impact its environment (e.g., electromagnetic fields).

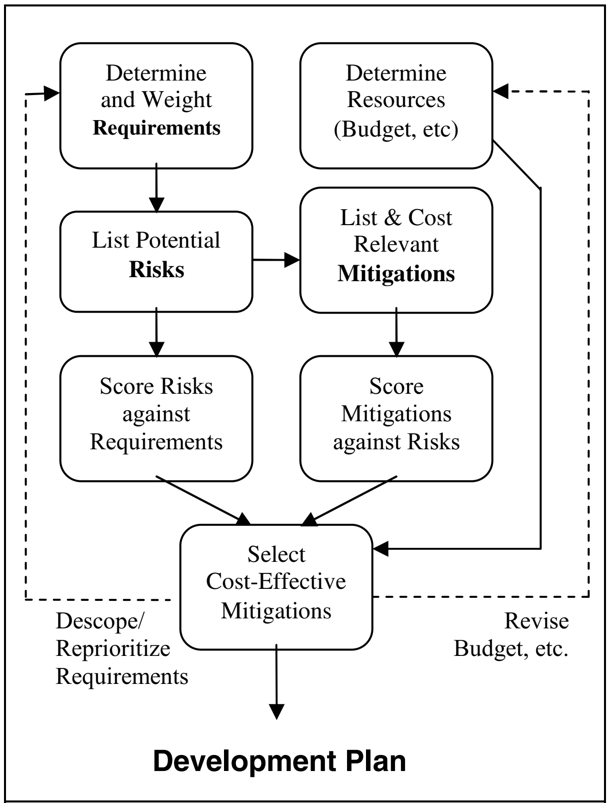

Determine and Weight Requirements
Determine Resources (Budget, etc)
List Potential Risks
List & Cost Relevant Mitigations
Score Risks against Requirements
Score Mitigations against Risks
Select Cost-Effective Mitigations
Descope/ Reprioritize Requirements
Revise Budget, etc.
Development Plan

Figure 2: DDP Process [6]

Risks:

- Development problems (inability to construct, test, repair, operate and maintain the system)
- The multitude of ways the operating system can fail to meet requirements.

Mitigations:

- Preventative measures that can be employed to reduce the likelihood of problems arising in the first place (e.g., coding standards, training, use of qualified parts)
- Detections that can be employed to uncover the presence of problems prior to fielding and use of the system (e.g., inspections, reviews, analyses, tests).
- Alleviations that can be employed to reduce the severity of Risks (e.g., array bounds checking coupled with appropriate responses).

Typical DDP applications have involved 5 – 15 experts drawn from the disciplines of mission-science, project planning, software and hardware engineering, quality assurance, testing, risk management, etc.

DDP's particular strength is that it can combine inputs from this wide variety of disciplines. It uses its relatively simple risk-based quantitative model to do so. Certainly this model is incapable of capturing all the nuances of a complex design. However, for early decision making, it is more important to be able to make key choices, those that, if done correctly, will lead to significantly superior designs. By seeking to be all encompassing of the relevant areas of expertise, DDP is able to avoid pitfalls of too narrow a focus on just the areas that are understood in depth. The simplicity of the quantitative risk model means that all areas can formulate their concerns to at least a coarse level of fidelity, which is often all that is needed to make key decisions.

There is a straightforward stepwise process to building and using a DDP model. The steps are illustrated in figure 2. It has been typical to require at least 4 sessions of 3 to 4 hours each to gather the DDP information for a non-trivial technology. A facilitator is needed to direct these sessions. This must be someone who both understands the DDP process, and has a feel for the broad range of concerns that the study must deal with. The facilitator guides the elicitation and decision making steps. The DDP tool is run throughout the sessions, its screen projected and visible to all the participants. As information is gathered, it is entered into the tool in real time. Someone conversant with the DDP tool controls the tool, does data entry, switches between the various visual presentations, etc. In some studies, the same individual has acted as both facilitator and tool controller; in others, separate individuals have filled these two roles.

The DDP process has been applied to spacecraft hardware, software and systems (including both hardware and software). Engineers from the hardware field are used to thinking about risks of various technologies. These fields have much published data about component reliability. In the software field, less is known (or published) about typical software risks. To help in this area, we have preloaded a set of software risks into a DDP database. This is a taxonomy of software risks from the Software Engineering Institute. [1] With this initial seeding, the engineers and domain experts do not have to begin with a "blank slate." (They are, of course, not limited to these software risks. They can down-select from this set to just those that they deem applicable, and can add others easily.)

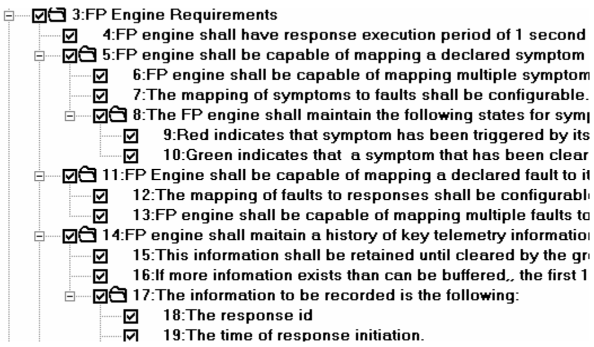
Custom software [2, 4] has been developed to support the DDP process. This software is used to:

- Capture on-the-fly the stakeholders' expressions of knowledge, and help organize this accumulation of knowledge.
- Perform calculations over the assembled information.
- Present back the information using a variety of cogent visualizations.
- Support the stakeholders in performing risk-informed decision making.

Examples of the DDP software's features are to be found in the next section.

## 5. Proof-of-concept software application

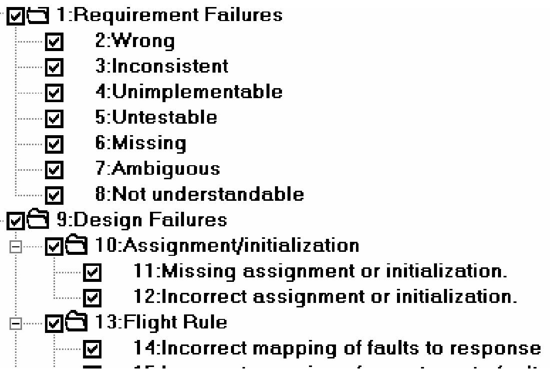
DDP has been used primarily on hardware components



```
3:FP Engine Requirements
    4:FP engine shall have response execution period of 1 second
    5:FP engine shall be capable of mapping a declared symptom
        6:FP engine shall be capable of mapping multiple symptom
        7:The mapping of symptoms to faults shall be configurable.
        8:The FP engine shall maintain the following states for sym
            9:Red indicates that symptom has been triggered by its
            10:Green indicates that  a symptom that has been clear
    11:FP Engine shall be capable of mapping a declared fault to i
        12:The mapping of faults to responses shall be configurabl
        13:FP engine shall be capable of mapping multiple faults to
    14:FP engine shall maitain a history of key telemetry informatio
        15:This information shall be retained until cleared by the gr
        16:If more infomation exists than can be buffered,, the first 1
        17:The information to be recorded is the following:
            18:The response id
            19:The time of response initiation.
```

**Figure 3. A portion of the requirements tree**

of systems, most often to study their advancement from a working prototype to engineering model. In this paper we present a proof-of-concept of how DDP could also be applied to strategic decision making during software development activities. It is our assertion that strategies for software development are analogous those for



```
1:Requirement Failures
    2:Wrong
    3:Inconsistent
    4:Unimplementable
    5:Untestable
    6:Missing
    7:Ambiguous
    8:Not understandable
9:Design Failures
    10:Assignment/initialization
        11:Missing assignment or initialization.
        12:Incorrect assignment or initialization.
    13:Flight Rule
        14:Incorrect mapping of faults to response
```

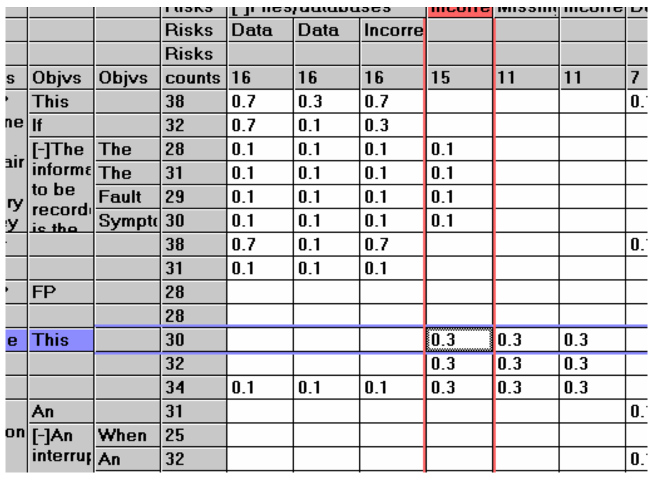
**Figure 4. A portion of the risks tree**

hardware system development. That is, we believe that the DDP model based on requirements, risks, and mitigations can be effective in strategic decision-making for software systems also.

Our case study for this assertion is a fault protection subsystem (in order to conceal proprietary information we present only some of its information). First, we cast the system requirements as DDP requirements. A portion of the resulting tree structure is seen in Figure 3 above.

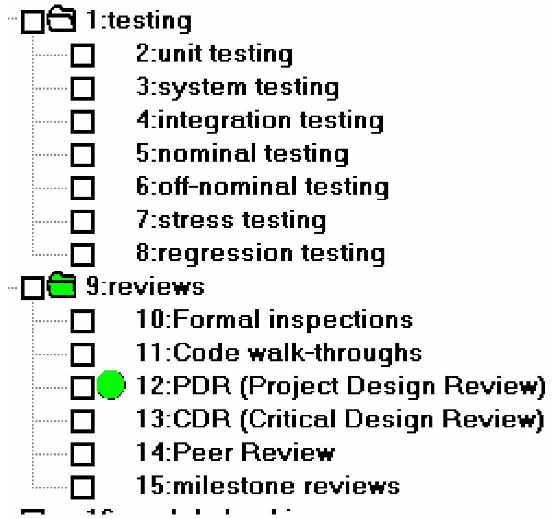
The risk tree is similar: a portion is seen in Figure 4.

Requirements, risks and mitigations are linked through two matrices. The first of these matrices correlates risks and requirements. (See Figure 5.) Engineers and domain experts are asked to estimate the impact that each risk, should it occur, would have on each requirement. Impacts are expressed as numbers in the range 0 to 1. An estimate of 1 means that this risk is so serious that, if it occurs, that requirement would be completely lost. A value of 0 implies that this risk will not effect this requirement at all. A value of ρ (0 < ρ < 1) means that the experts expect that risk to cause the proportion ρ of that requirement to be lost. It is also possible to provide a non-numeric value, which will be ignored in the calculations performed by the DDP tool, but which has utility as a placeholder (e.g., a "TBD" value which someone will be responsible for looking up). For the purpose of our proof-of-concept demonstration, we populated this matrix with our own estimates. If this were a real study, we would follow the usual process and convene a group of experts who together would provide and vet such information.



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Risks | [-]Files/databases | | | Incorre | Missin | Incorre | D |
| | | | Risks | Data | Data | Incorre | | | | |
| | | | Risks | | | | | | | |
| s | Objvs | Objvs | counts | 16 | 16 | 16 | 15 | 11 | 11 | 7 |
| ' | This | | 38 | 0.7 | 0.3 | 0.7 | | | | 0. |
| ne | If | | 32 | 0.7 | 0.1 | 0.3 | | | | |
| air | [-]The | The | 28 | 0.1 | 0.1 | 0.1 | 0.1 | | | |
| | informa | The | 31 | 0.1 | 0.1 | 0.1 | 0.1 | | | |
| ry | to be | Fault | 29 | 0.1 | 0.1 | 0.1 | 0.1 | | | |
| y | record is the | Sympt | 30 | 0.1 | 0.1 | 0.1 | 0.1 | | | |
| ' | | | 38 | 0.7 | 0.1 | 0.7 | | | | 0. |
| | | | 31 | 0.1 | 0.1 | 0.1 | | | | |
| ' | FP | | 28 | | | | | | | |
| | | | 28 | | | | | | | |
| e | This | | 30 | | | | 0.3 | 0.3 | 0.3 | |
| | | | 32 | | | | 0.3 | 0.3 | 0.3 | |
| | | | 34 | 0.1 | 0.1 | 0.1 | 0.3 | 0.3 | 0.3 | |
| | An | | 31 | | | | | | | 0. |
| on | [-]An | When | 25 | | | | | | | |
| | interrup | An | 32 | | | | | | | 0. |

**Figure 5. A portion of the Requirements (rows) x Risks (columns) matrix**



- 1:testing
  - 2:unit testing
  - 3:system testing
  - 4:integration testing
  - 5:nominal testing
  - 6:off-nominal testing
  - 7:stress testing
  - 8:regression testing
- 9:reviews
  - 10:Formal inspections
  - 11:Code walk-throughs
  - ● 12:PDR (Project Design Review)
  - 13:CDR (Critical Design Review)
  - 14:Peer Review
  - 15:milestone reviews

**Figure 6. Partial tree of software risk mitigations**

In this study we have so far worked with only a subset of the wide range of activities applicable to the reduction of software risk. Figure 6 shows (most of) our tree structure of such activities:

The second matrix (not shown here) correlates risk and mitigations. That is, for each mitigation-risk pair, domain experts are asked to estimate the ability of that mitigation to alleviate, detect or prevent that risk. Again, these are numbers in the range 0 to 1, with 1 meaning that this mitigation completely eliminates this risk; 0 means that it has no effect on this risk. (For mitigations that detect risks, the intent is that these be applied before the spacecraft is launched, in time for any detected problems to be repaired).

## 5.1 Decision Making

Up to this point in the DDP process, the task of the DDP tool is to record these decisions. After this data is collected, the focus of the process switches to one of decision-making, guided by the recorded information. The primary purpose of DDP has been to help guide the selection of which of the mitigating actions to take to overcome the risk and therefore to achieve the requirements. Since there are typically many more possible mitigations than can be simultaneously afforded, the aim of this step is to emerge with a cost-effective selection from among them. Another outcome can be the strategic decision to modify and/or abandon some requirements if it becomes clear that it is not possible to

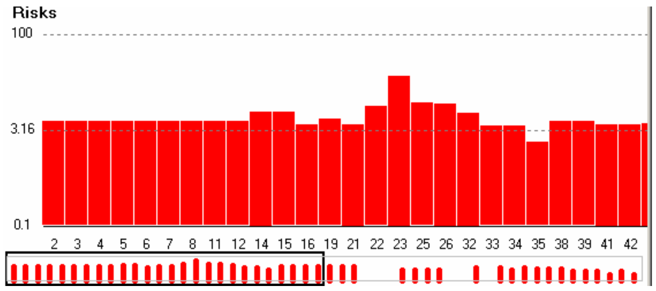


**Figure 7. Bar chart of (unmitigated) risks**

satisfactorily achieve them all with the resources available. This is called descoping [5].

In addition to the matrix and list views used primarily for input of data, DDP provides several views appropriate for study of various measures calculated from this data. One of these is a familiar bar chart view. For example, when showing the aggregate impact that each risk has on the requirements (taking in to account both the weight of the requirements and the strengths of the impacts), this is useful to indicate the "tall pole" risks worthy of further attention. See Figure 7.

The heights of the bars indicate the sum total impacts (on requirements) of each of the risks. These can be sorted in various ways; the most commonly used one being in descending order of magnitude. The small strip along the bottom shows a thumbnail view of the entire bar

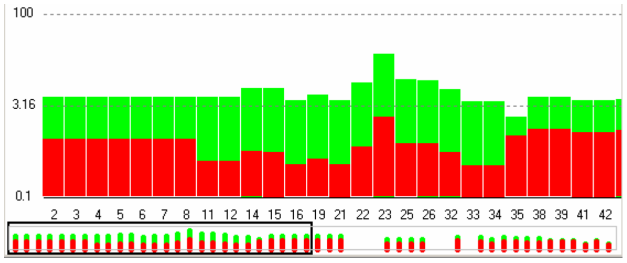


**Figure 8. Bar chart of mitigated risks**

chart. The black rectangle overlaying that thumbnail shows which portion of the entire bar chart is currently visible at full scale.

The tallest bars are those risks that have the greatest sum total impacts on weighted requirements – these are the risks that one would want to focus attention on.

As mitigations are selected, their effects reduce risks (by decreasing the likelihoods and/or impacts of those risks, depending on the kind of mitigation). The same bar chart is used to reveal this – see Figure 8. Green shows the risk levels *before* mitigation, red shows the risk levels *after* mitigation.

The advantage of views such as these is that they allow the users to immediately and dramatically see the risk-reducing effects of mitigations. One of the banes of software quality assurance is convincing developers of the value of the activities it recommends. The aim of this work is to reveal the benefits of assurance activities in terms of the risks they reduce. Developers can still exercise choice over which mitigation activities they wish to perform, but now their choices are informed by the risk implications that stem from omitting some activity. DDP offers further assistance in this regard by using this same bar chart display to show *comparisons* among alternate selections. This is shown in Figure 9, where an alternate selection of mitigations has been made. Black shows where risks have *increased* relative to the first selection; *yellow* shows where risks have decreased.

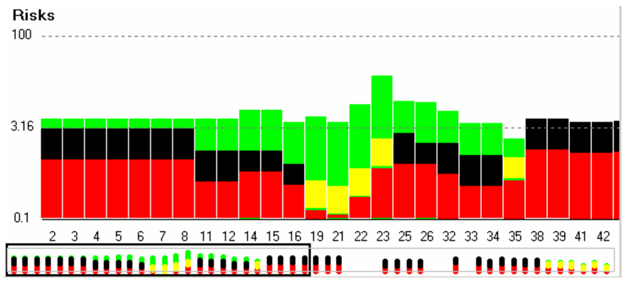


**Figure 9. Comparison of alternate mitigation selections**

Another display of risk status offered by DDP is the familiar 2-D risk chart, the axes of which are likelihood and impact. This is seen in Figure 10. The red-yellow-green colored regions demark high, medium and low risk areas (because the axes of the chart are log scale, the diagonal boundaries between these regions are themselves contours of constant risk). It is a common policy to determine the managerial treatment of risks depending on which of these regions they fall within (the boundaries between regions being another strategic decision). For example, high likelihood high impact risks are brought to the attention of the project as a whole, are more closely tracked over time, etc. Other risk tools commonly use a rectangular matrix of risk regions for a similar purpose. DDP can generate these other chart forms too.

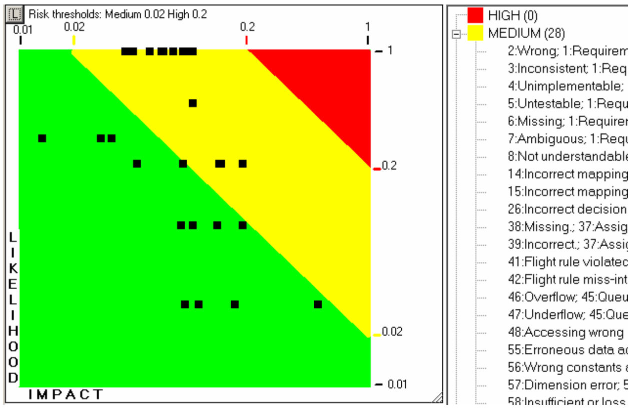


**Figure 10. 2-D risk chart**

A selection of mitigations has a benefit (the net effect is to reduce risks, and thereby improve the expected attainment of requirements), but also has a cost (the sum of the costs of performing the mitigations themselves, and for those problem-detection type mitigations, such as tests and analyses, the costs of correcting the defects they discover). DDP automatically computes both the benefit and cost of the currently selected mitigations. So far the illustrations have concentrated on showing the benefit side of the equation. DDP also displays a small but important cost meter: as mitigations are selected, the $ figure on the cost meter increases. In selecting mitigations, users must take into account both the benefits and the costs. Emerging with a cost-effective set of mitigations is one of the primary uses of DDP.

To assist in this DDP has a built-in mechanism for heuristic optimization. The current distribution of DDP uses simulated annealing for this purpose, but we have also experimented successfully using genetic algorithms [8]. Optimization can be set to search for the selection of mitigations that will achieve maximum requirements attainment while staying at or below some cost ceiling, to search for the selection of mitigations that minimizes the cost of attaining (or improving on) some prescribed lower bound of requirements attainment, or to search for some hybrid combination of the two. The result of a search is the identification of (near) optimal mitigation selections. These selections can be scrutinized through DDP's various displays just as can a manually chosen selection of mitigations, so users can employ the heuristic search to reveal promising solutions, and thereafter fine-tune those solutions as needed (e.g., they might have reason to prefer one slightly inferior solution over another slightly superior solution, because it avoids the use of a scarce testing resource, say). The heuristic search can also be repeated at intervals across the cost spectrum to reveal the overall shape of the cost/benefit space. Our software dataset is still only partially completed with respect to its list of mitigations, so it is premature to utilize this capability just yet. As our study progresses, we look forward to experimenting with this.

## 6. Discussion

There are some strengths and weaknesses to the DDP model and how it is applied. We discuss these in relationship to related work on software assessment.

### 6.1 Strengths

The relative simplicity of the DDP model gives it the ability to span a wide range of concerns (e.g., technical and programmatic risks). In this respect it is reminiscent of, for example, the openness of the i* model [9, 10], used for evaluation of (among other things) software design alternatives.

The information that goes into a DDP model can be a mixture of experts' estimates and (where available) historical knowledge.

The data that comprises the model is extensible. Information can be added, changed and removed on-the-fly. For example, risks specific to the case at hand can be introduced, irrelevant ones removed, and pre-populated data adjusted (e.g., if the analysis team includes an expert on the use of formal methods, the cost for such analysis might be reduced, and its effectiveness increased).

The data to populate a DDP model, while voluminous, is straightforward, and experience has shown the feasibility of capturing sufficient information (sufficient to emerge with insightful findings) with modest amounts of experts' time. For our technology infusion studies, it is typical to require the involvement of, say, 10 experts in 4 half-day sessions. The total amount of effort is thus on the order of 160 hours. When key decisions are being made as to whether to commit to a multi-million-dollar next step in the advancement of a technology, such an investment of time is not unreasonable.

Overall DDP is reminiscent of QFD (Quality Function Deployment) method [11], widely used across a range of industries and application areas. DDP has a more quantitative, risk-centric perspective, with a probabilistic interpretation pervading its Requirements-Risks-Mitigations model.

### 6.2 Weaknesses

The DDP model schema (notably the formulae by which impacts of multiple Risks on the same Requirement "add up", and by which multiple Mitigations against the same Risk combine) is inflexible. These formulae are pre-set, and do not necessarily apply well to all situations. While there are some workarounds that can be employed if need be (e.g., representing a combination of mitigations as a distinct mitigation whose effectiveness at reducing risks can be asserted), they are clumsy to use. Other researchers adopt models that can be constructed to match the case at hand, and thus more faithfully represent the software development process, e.g., the Bayesian Belief Net models of [12], or the simulation models of [13].

DDP lacks a means for validation of its models. The aforementioned formulae were chosen to be plausible, but are not based on a solid body of evidence. Likewise the experts' estimates that comprise a significant portion of most DDP models are constructed on-the-fly, and so do not have an explicit pedigree to experiential data. This is in contrast to software estimation techniques such as COCOMO and, more recently, COQUALMO and iDave [14, 15] which are derived from data from past software projects, possibly tempered by a consensus process of experts (e.g., using Delphi techniques). Also similar is the stochastic model of [16]

There is still an "art" to populating a DDP model. The use of pre-populated models as a starting point (e.g., a risk taxonomy of generic software development problems) is somewhat helpful, both to establish an overall structure, and to serve as a reminder (much like a checklist). However, there is need for discretion over how much additional information to add (what is the scope of the study?), and over how much detail to descend to (e.g., does the single Mitigation "software inspections" suffice, or is there need to distinguish among alternative forms of inspections, e.g., Fagan inspections, Perspective Based Reading, etc.). The danger of staying at too narrow a scope and/or too high a level is lack of coverage and discrimination among significantly distinct cases, while the danger of overly broadening the scope and/or descending to too low a level is the increased effort it takes to populate the model. We address this problem by using a DDP-knowledgeable person to facilitate the meeting (in fact, we usually use two such people – one to serve as facilitator, the other to "drive" the DDP software).

DDP's model probabilistic model is overly simplistic when compared to the structures seen in use in full-fledged Probabilistic Risk Assessment [17]. For example, PRA tools such as Sapphire, QRAS and Galileo have explicit notions of temporal dependencies (through event sequence diagrams or phase-dependent fault tree gates), and of probability distributions with which to capture and reason about uncertainties. Although we have recently incorporated logical fault trees into DDP's Risks structures [18], these are but a small step toward the full representational power of PRA. We have explored the use of DDP as an agile precursor step that serves to indicate where a more elaborate PRA model needs to be constructed [19]. In the software arena, however, it is less routine to apply PRA methods; instead, modest applications of fault trees, Software FMECAs and hazard analyses (and combinations of them, e.g., [20]) are the norm. For such applications DDP seems to be relatively well suited.

In the software arena, DDP is reminiscent of the KAOS models of [21], specifically their treatment of "obstacles" (akin to DDP's "Risks"). However, whereas the KAOS models emphasize a logic-structure based treatment (including temporal logic to capture, e.g., timing requirements), DDP has emphasized instead a quantitative treatment more immediately suited to capture and tradeoff reasoning over and between so-called "non-functional requirements". It is intriguing that in recent work the KAOS approach is being extended to incorporate quantitative reasoning [22], while, as mentioned above, logical fault tree constructs are being incorporated into DDP. It seems these alternate approaches are each expanding in the direction of the other.

## 7. Conclusions

The paper has described the DDP process, and shown aspects of our ongoing study of software specific application of our risk-informed decision making process and its software support, DDP.

The DDP process and tool has proven itself useful in strategic planning for hardware-based technology at JPL over the past few years. In many of the applications, the DDP process has yielded insights that the experts involved describe as both *surprising* and *correct* – surprising in the sense that they would not have emerged with those insights until significantly later in the development process, by which time reacting to them would be more costly; and correct in the sense that when they trace through the DDP data underpinning the insights they agree upon its validity. Examples of kinds of insights include recognition of a risk that is more serious than would have been suspected (and so warrants increased efforts to mitigate it, or, in some cases, revision of the requirements), and recognition that a hitherto unfavored mitigation is really needed (or in some cases the reverse – that a regularly favored mitigation is in this instance unnecessary).

We ascribe its benefit to the following factors:

- It is typical for DDP studies to involve a non-trivial amount of data. There are commonly dozens (or even hundreds) each of requirements, risks and mitigations, and there may well be thousands of non-zero impact and effect values connecting these.
- The volume and interconnectedness of this data reflects the complexity of the challenges inherent in the design of complex systems. DDP aims to help in this design process, and so must manage this quantity of information in such a way as to allow the engineers and domain experts both to provide the information, and make decisions on the basis of that aggregated information.

In our study so far of application of DDP to a software system, we find these same phenomena recurring. This gives us confidence that DDP will prove to be useful for strategic decision making in software development.

## 8. References

[1] Carr, M. J., S. L. Konda, et al. (1993). Taxonomy-Based Risk Identification. Pittsburg, PA, Software Engineering Institute: 78.

[2] Cornford, S. L., Feather, M.S. et al. (2001). DDP – A tool for life-cycle risk management. *IEEE Aerospace Conference*, Big Sky, Montana, 2001.

[3] Feather, M.S. & Cornford, S.L., "Quantitative Risk-Based Requirements Reasoning", *Requirements Engineering* (2003) 8: 248-263, Springer-Verlag, London.

[4] Feather, M.S., Cornford, S.L. Dunphy, J. & Hicks, K.A. (2002). A Quantitative Risk Model for Early Lifecycle Decision Making; *Proceedings of the Conference on Integrated Design and Process Technology*, Pasadena, California, June 2002. Society for Design and Process Science.

[5] Feather, M.S., S.L. Cornford & K.A. Hicks (2002) Descoping; Proceedings *of the 27th IEEE/NASA Software Engineering Workshop*, Greenbelt, Maryland, Dec 2002. IEEE Computer Society.

[6] Kiper, J.D. and Feather, M.S. "From Requirements through Risks to Software Architecture for Plan-based and Agile Processes", *Proceedings of the Workshop on Requirements Engineering for Adaptive Architectures*; Monterey Bay, CA, Sept. 2003.

[7] Lutz, R. and Mikulski, C. "Empirical Analysis of Safety-Critical Anomalies During Operations", *IEEE Transactions on Software Engineering*, 30(3), March, 2004

[8] Cornford, S.L., M.S. Feather, J. Dunphy, J. Salcedo & T. Menzies, 2002, "Optimizing the Design of end-to-end Spacecraft Systems using Risk as a Currency", *IEEE Aerospace Conference*, Big Sky, Montana, Mar 2003, pp. 7.3361 – 7.3368.

[9] Chung, L., B.A. Nixon, B.A., E. Yu, E., and Mylopoulos, J., Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, Boston, 1999.

[10] Mylopoulos, J., Chung, L., Liao, S., Wang, H., and Yu, E., 2001, "Exploring Alternatives during Requirements Analysis", *IEEE Software* 18(1): 92-96.

[11] Akao, Y. 1990 "Quality Function Deployment", Productivity Press, Cambridge, Massachusetts.

[12] Fenton, N., Marsh, W., Neil, M., Cates, P., Forey, S., and Tailor, M.; Making resource decisions for software projects. *Proceedings of the 26th International Conference on Software Engineering*, 2004, May 23-28, 2004, pp. 397 – 406.

[13] Wakeland, W., Martin, R.H. & Raffo, D. "Using Design of Experiments, Sensitivity Analysis, and Hybrid Simulation to Evaluate Changes to a Software Development Process: A Case Study", *Proceedings of International Workshop on Software Process Simulation and Modeling (ProSim'03)*, Portland, OR, May 2003

[14] Boehm, B., et al., Software Cost Estimation with COCOMO II, Prentice Hall, Upper Saddle River, NJ, 2000.

[15] Boehm, B., Huang, L., Jain, A. & Madachy, R., "The ROI of Software Dependability: The iDave Model", *IEEE Software*, 12(3): 54-61, May/June2004.

[16] Stutzke, M.A. and Smidts, C.S., "A Stochastic Model of Fault Introduction & Removal During Software Development", *IEEE Transactions on Reliability*, 50(2): 184-193, June 2001.

[17] Stamatelatos, M., et al. "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners" http://www.hq.nasa.gov/office/codeq/doctree/praguide.pdf Office of Safety and Mission Assurance, NASA Headquarters, Washington, DC 20546, August 2002.

[18] Feather, M.S., Towards a Unified Approach to the Representation of, and Reasoning with, Probabilistic Risk Information about Software and its System Interface", to appear in the *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering*, Saint Malo, Bretagne, France, 2-5 November 2004. Available from: http://eis.jpl.nasa.gov/~mfeather/Publications.html

[19] Cornford, S.L., Paulos, T., Meshkat, L. & Feather, M., "Towards More Accurate Life Cycle Risk Management Through Integration of DDP and PRA", *IEEE Aerospace Conference*, Big Sky, MT, March 2003, pp. 2.1106-2.1200.

[20] Lutz, R. and Woodhouse, R. "Requirements Analysis using Forward and Backward Search", *Annals of Software Engineering, Special Volume on Requirements Engineering*, 3, pp. 459-475, 1997.

[21] A. van Lamsweerde & E. Letier, "Integrating Obstacles in Goal-Driven Requirements Engineering", *ICSE98 – 20th International Conference on Software Engineering*, IEEE-ACM, Kyoto, April 1998.

[22] Letier, E. & van Lamsweerde, A., "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering", to appear in the *Proceedings of ACM/SIGSOFT 2004/FSE-12*, Newport Beach, CA, 2004.